Overview

- Why Learn Python
- Getting started – setting up your environment
- Python – Hello World
- Python – using numbers
- Python – Getting user input

Why Learn Python

- Python is an easy to learn but very powerful language.
- Python can be used creatively in areas such as
    - robotics
    - artificial intelligence
    - physical computing
    - web interaction
    - games development
- It is also used as a scripting language in programs such as Cinema 4D and Unreal Engine.
- Perhaps the easiest of the programming languages to learn,.
- Python is a good language for someone beginning to code.

However!

- Learn to walk before you can run!

Setting up your environment

- Thonny is a free Python Integrated Development Environment (IDE).
- It has a built-in debugger that can help when you run into bugs in your code.

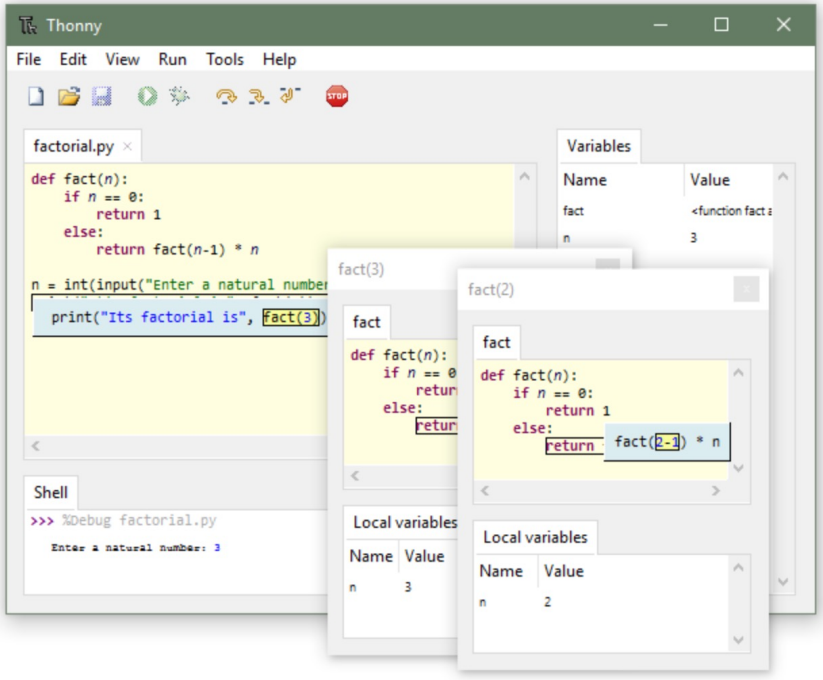
- https://thonny.org/

Thonny
Python IDE for beginners

Download version 3.3.5 for
Windows • Mac • Linux

NB! Windows installer is signed with new identity and you may receive a warning dialog from Defender until it gains more reputation.

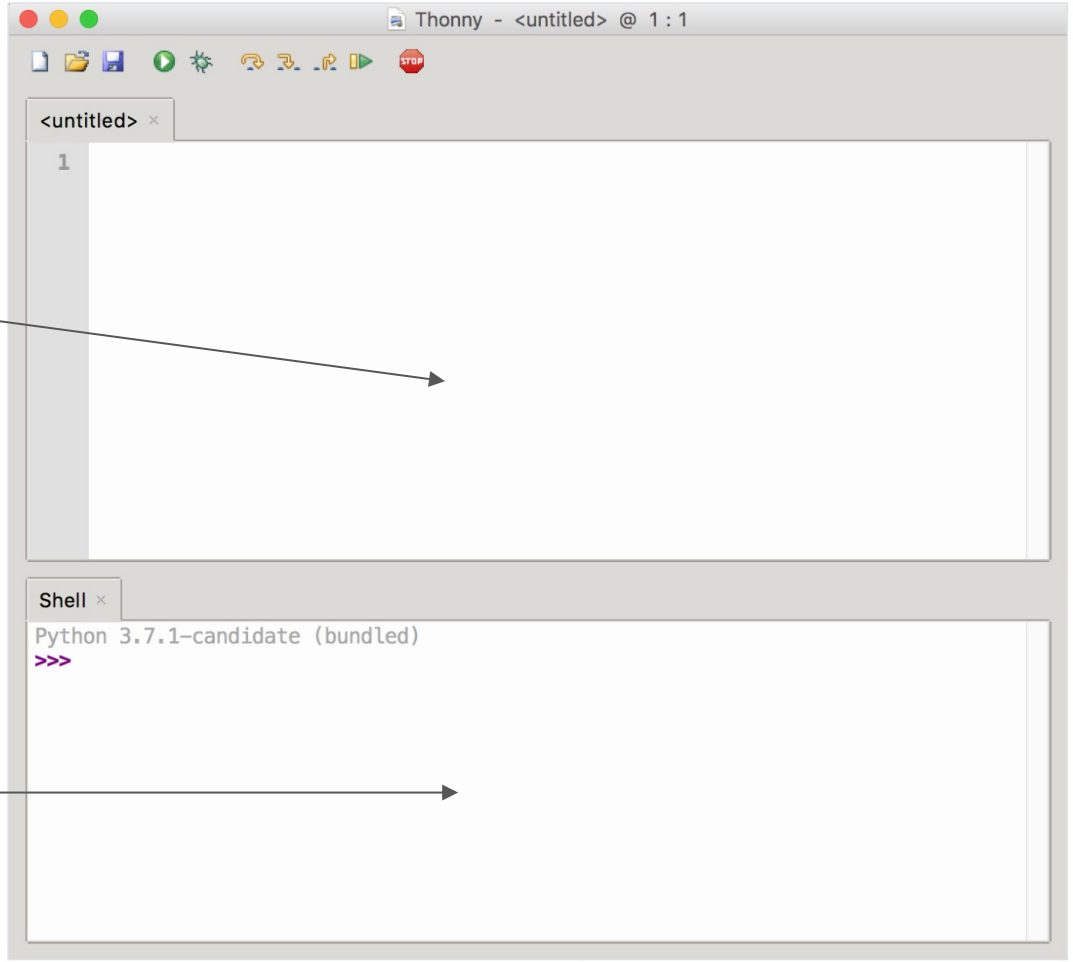
Just click "More info" and "Run anyway".



IDE

Code Editor

Shell



- **A:** Create a new file.
- **B:** Open a file that already exists on your computer.
- **C:** Save your code.
- **D:** Run your code.
- **E:** The bug icon allows you to debug your code.
- **F-H:** The arrow icons allow you to run your programs step by step.

F take a big step - jumping to the next line or block of code.
G take a small step - diving deep into each component of an expression.
H arrow tells Python to exit out of the debugger.
I: The resume play mode from debug mode.
J: The stop icon allows you to stop running your code.

Hello World!
– Your first Python code

1) In the Code editor type:

print (“Hello World”)

2) Click on the play button

3) Output to Shell

Concept - Strings

- Computer programmers refer to blocks of text as *strings*.
- In our last exercise, we created the string “Hello world!”.
- In Python a string is either surrounded by double quotes ("Hello world") or single quotes ('Hello world’).
- It doesn’t matter which kind you use, just be consistent.

Exercise

Print your name using the print() command.

Concept - Variables

- A variable is a way of naming and storing a value for use by the program
- If there is a greeting we want to present, a date we need to reuse, or a user ID we need to remember we can create a ***variable*** which can store a value.
- In Python, we ***assign*** variables by using the equals sign **(=)**.
- In the example, we store the string "**Hello World**" in a **variable** called **message**.

Bye.py

```
1
2
3 message = "Hello World"
4 print (message)
5
6
7
8
```

Shell

```
>>> %Run Bye.py
  Hello World
>>>
```

Variables

```
1  meal = "Porridge"
2
3  print("Breakfast:")
4  print(meal)
5
6  meal = "Soup"
7
8  print("Lunch:")
9  print(meal)
10
11 meal = "Pizza"
12
13 print("Dinner:")
14 print(meal)
15
16
```

- It's no coincidence we call these "variables".
- We can update the content of a variable as the program runs
- Variables can't have spaces or symbols in their names other than an underscore (_). They can't begin with numbers but they can have numbers after the first letter (e.g., cool_variable_5).

Bugs

- Humans are prone to making mistakes.
- Humans are also typically in charge of creating computer programs.
- To compensate, programming languages attempt to understand and explain mistakes made in their programs.
- Python refers to these mistakes as *errors* and will point to the location where an error occurred with a ^ character.
- When programs throw errors that we didn't expect to encounter we call those errors *bugs*.

SyntaxError

- **SyntaxError** means there is something wrong with the way your program is written
- punctuation that does not belong
- a command where it is not expected
- a missing parenthesis

helo.py

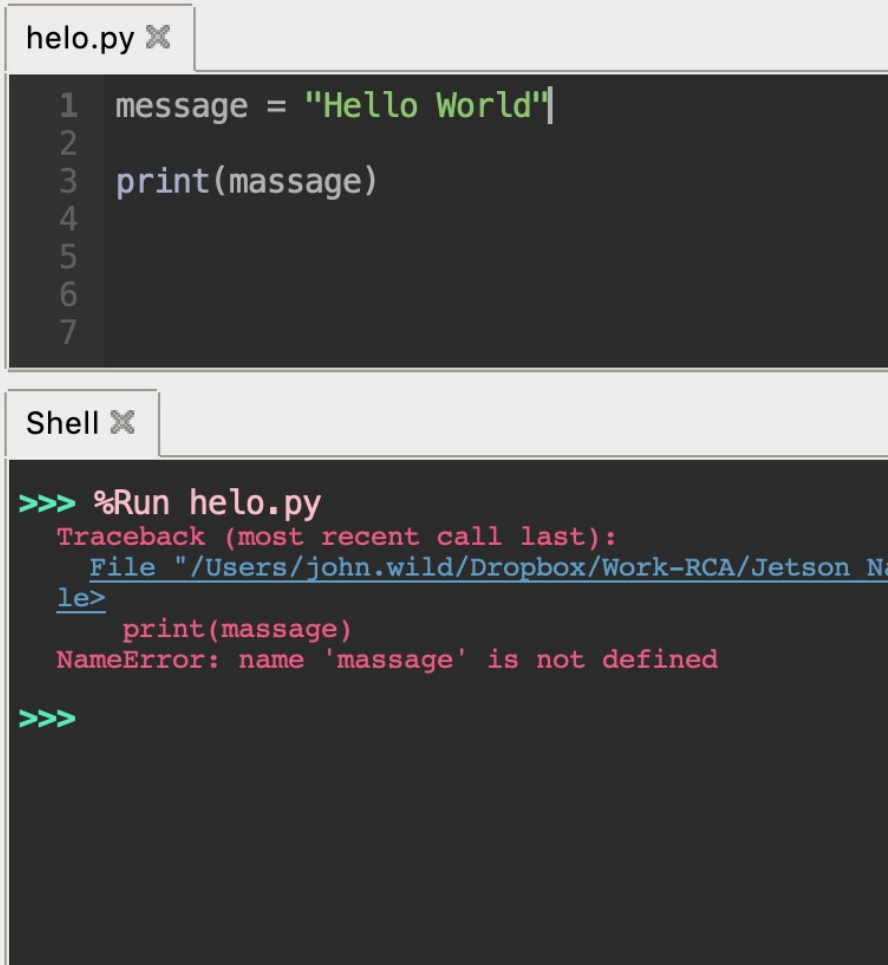
```
1  message = "Hello World'
2
3  print(message)
4
5
6
7
```

Shell

```
>>> %Run helo.py
  Traceback (most recent call last):
    File "/Users/john.wild/Dropbox/Work-RCA/Jetson Na
      message = "Hello World'
                              ^
  SyntaxError: EOL while scanning string literal

>>>
```

NameError

- A **NameError** occurs when the Python interpreter sees a word it does not recognize.
- Code that contains something that looks like a variable but was never defined will throw a NameError.



Next Up!

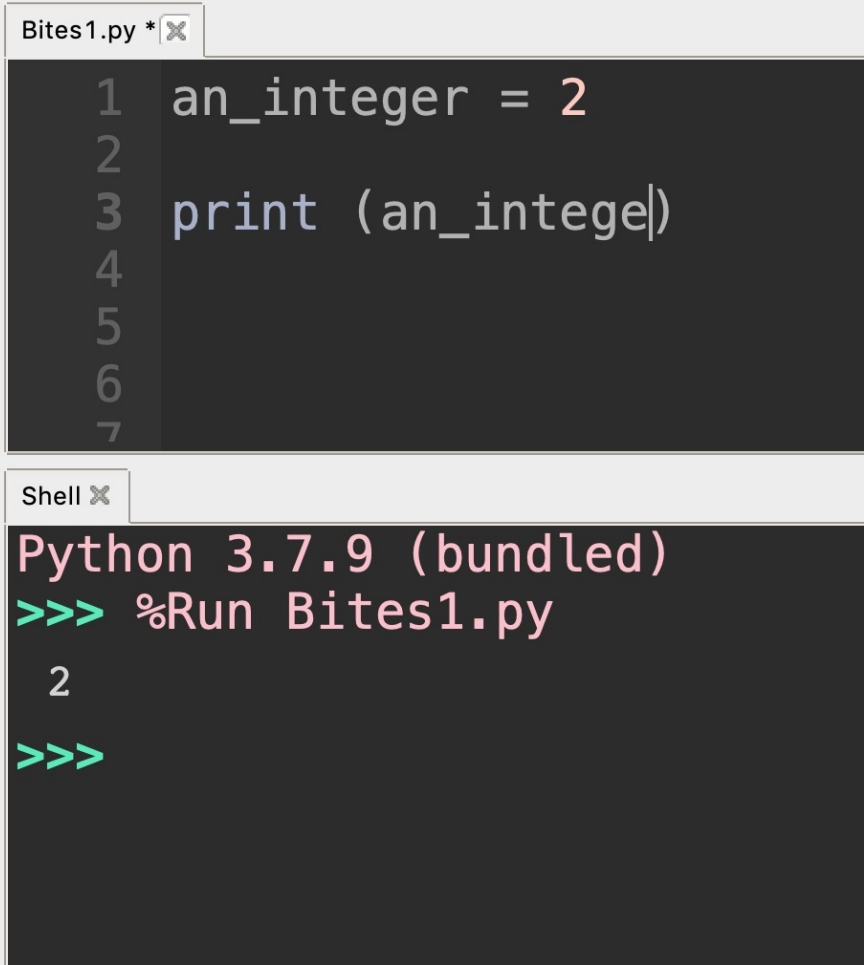
- Numbers
- Integer
- floating-point number
- Calculations
- Variables and Numbers
- Commenting your Code

Numbers

- Computers can understand much more than just strings of text.
- Python has a few numeric ***data types***.
- It has multiple ways of storing numbers.
- Which one you use depends on your intended purpose for the number you are saving.

Concept - Integer

- An *integer*, or int, is a whole number.
- It has no decimal point and contains all counting numbers (1, 2, 3, ...) as well as their negative counterparts and the number 0.
- If you were counting the number of people in a room, the number of jellybeans in a jar, or the number of keys on a keyboard you would likely use an integer.



Concept - floating-point number

- A *floating-point number*, or a float, is a decimal number.
- It can be used to represent fractional quantities as well as precise measurements.
- If you were measuring the length of your bedroom wall you would likely use a float. i.e 2.32 meters

Bites1.py

```
1  a_float = 2.5
2
3  print (a_float)
4
```

Shell

```
>>> %Run Bites1.py
  2.5
>>>
```

Calculations

Function	Python
Addition	+
Subtraction	-
Multiplication	*
Integer Division	//
Float Division	/

Bites1.py *

```
1
2  print (5 + 5)
3
4  print (5 - 5)
5
6  print (5 * 5)
7
8  print (5 // 5)
9
10 print (5 / 5)
11
12
13
```

Shell

```
>>> %Run Bites1.py

  10
  0
  25
  1
  1.0

>>>
```

Variables and Numbers

- Variables that are assigned numeric values can be treated the same as the numbers themselves.
- Performing arithmetic on variables does not change the variable — you can only update a variable using the = sign.

Bites1.py

```
1
2  coffee_price = 1.50
3  number_of_coffees = 4
4
5  print(coffee_price * number_of_coffees)
6
7  print(coffee_price)
8
9  print(number_of_coffees)
10
11 coffee_price = 2.00
12
13 print (coffee_price * number_of_coffees)
14
15 print(coffee_price)
16
17 print (number_of_coffees)
```

Shell

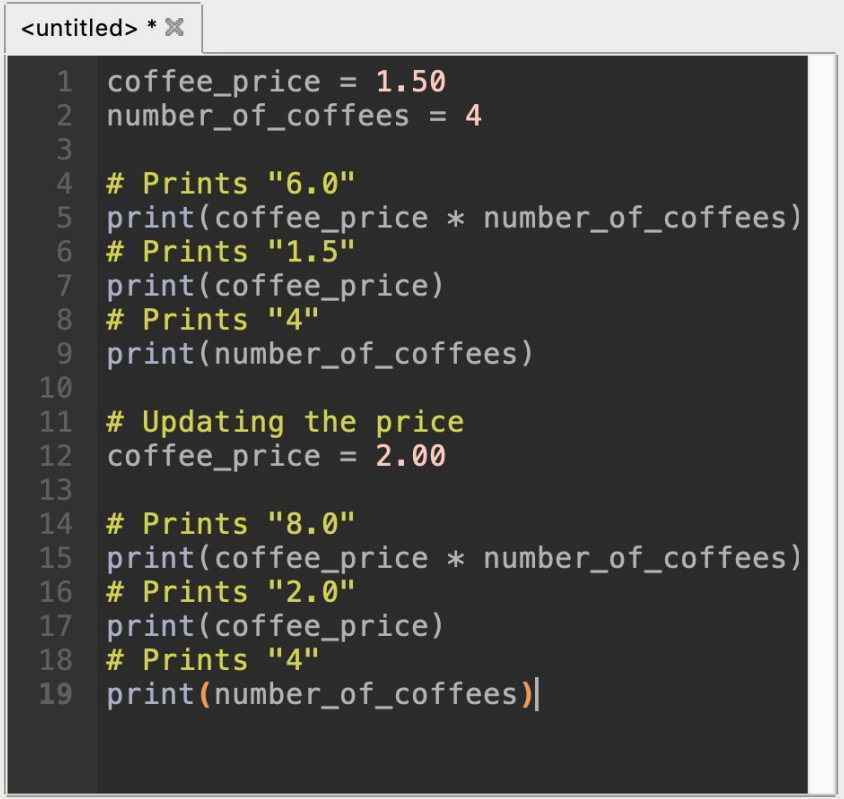
```
>>> %Run Bites1.py

  6.0
  1.5
  4
  8.0
  2.0
  4

>>>
```

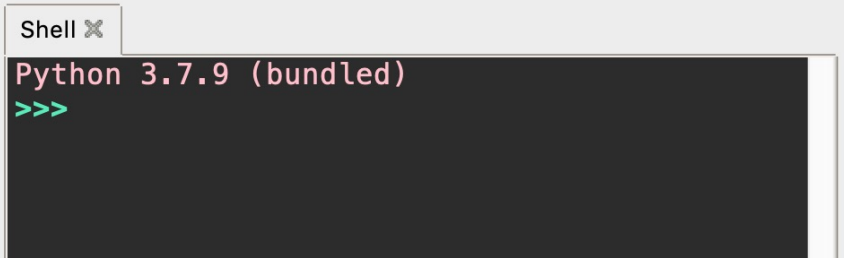
Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

Creating a Comment

- Comments starts with a **#**
- Python will ignore them everything on the line after the **#**





Take a 5min break

Concatenate

If you want to concatenate a string with a number you can

1) make the number a string using the str() function.

OR

2) print(string1, number, string

ten.py | test.py | Run current script

```
1  birthday_string = "I am "
2  age = 10
3  birthday_string_2 = " years old today!"
4
5  # Concatenating an integer with strings using str()
6  full_birthday_string = birthday_string + str(age) + birthday_string_2
7
8  print(full_birthday_string)
9
10 # This also prints "I am 10 years old today!"
11 print(birthday_string, age, birthday_string_2)
```

Shell

```
>>> %Run test.py

  I am 10 years old today!
  I am  10  years old today!

>>>
```

Task!

1) Create two string variables
2) Create one integer variable
3) Create a new string combining all three by converting the number to a string using str()
4) Test using the , method
5) Print all to the shell

Plus Equals (+=)

- Python offers a shorthand for updating variables.
- When you have a number saved in a variable and want to add to the current value of the variable, you can use the += (plus-equals) operator.
- variable1 += variable2

ten.py | test.py | Run current script

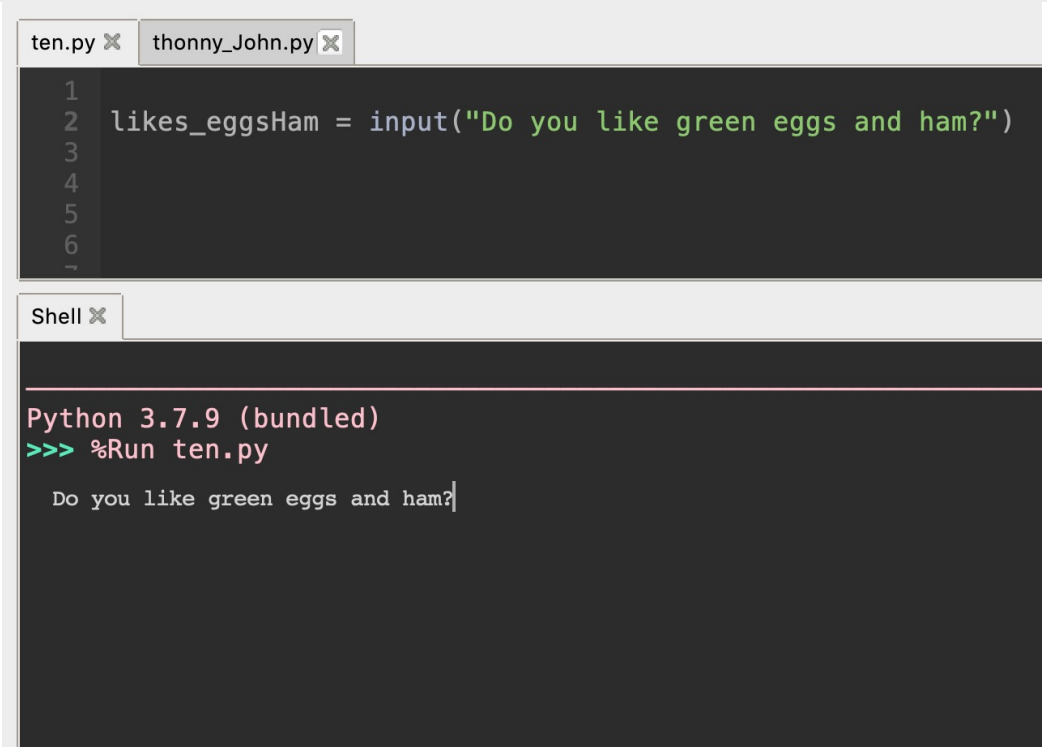
```
1  total_price = 0
2
3  new_sneakers = 50.00
4
5  total_price += new_sneakers
6
7  nice_sweater = 39.00
8  fun_books = 20.00
9  # Update total_price here:
10 total_price += nice_sweater
11 total_price += fun_books
12 print("The total price is", total_price)
```

Shell

```
>>> %Run test.py
  The total price is 109.0
>>>
```

Concept - Input

- We often want a user of a program to enter new information into the program.
- How can we do this? As it turns out, another way to assign a value to a variable is through user **input**.
- While we output a variable's value using print(), we assign information to a variable using input().
- The input() function requires a prompt message, which it will print out for the user before they enter the new information. For example:



Concept - Input

```
likes_eggsHam = input("Do you like green eggs and ham?")
```

- In the example above, the following would occur:
- The program would print " Do you like green eggs and ham? " for the user.
- The user would enter an answer (e.g., ”I do") and press **enter**.
- The variable **likes_eggsHam** would be assigned a value of the user’s answer.
- By default user input is a string

Input - numbers

```
string = input("Input some text ")
```

- By default user input is a string
- If you want int wrap input with int()

```
integer = int(input("Input a whole number "))
```

- If you want a float wrap input with float()

```
Float = float(input("Input a fraction "))
```

Task – 15min

- Create variables containing the price of different types of coffee

  - Cappuccino = 2.5
  - Flat white = 2.0
  - black = 1.5

- **Input** – ask the user to enter the quantity of each
- **Output** – print the total coast of the order

tes New

```
1  cappuccino = 2.5
2  flat_white = 2.0
3  black = 1.5
4
5  cups = int(input("How mmany cappuccinos?"))
6  total = cups * cappuccino
7  cups =  int(input("How mmany flat whites?"))
8  total += cups * flat_white
9  cups = int(input("How mmany cups of black coffee?"))
10 total += cups * black
11
12 print(total)
13
14
15
```

Shell

```
>>>
```

Error Catching

Problem

- cups = int(input("How many cappuccinos?"))
- Throws an error if the input is not an nummber

Solution

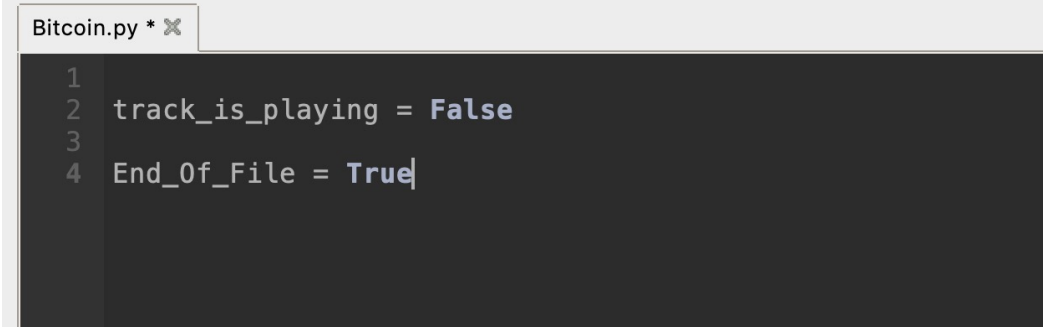
```
try:
    cups = int(input("How many cappuccinos?"))
except ValueError:
    print ("number not enterd")
    cups = 0
```

Data types

- Data types
- - **Strings**
  - **Integers**
  - **Floating Point Numbers**

  - **Bool**

Concept - Boolean



- A boolean expression is a statement that can either be True or False.
- True and False are their own special data type: bool.
- True and False are the only bool types, and any variable that is assigned one of these values is called a **boolean variable**.
- Boolean variables can be created in several ways. The easiest way is to simply assign True or False to a variable:

Equal Operator ==

- The equal operator, **==**, is used to compare two values, variables or expressions to determine if they are the same.
- If the values being compared are the same, the operator returns **True**, otherwise it returns **False**.
- The operator takes the data type into account when making the comparison, so a string value of "2" is *not* considered the same as a numeric value of 2.

bool.py

```
3  num1 = 6
4  num2 = 8
5
6  my_bool = num1 == num2
7
8  print("The result from 6 == 8 is",my_bool)
9
10 num2 = 6
11
12 my_bool = num1 == num2
13
14 print("The result from 6 == 6 is",my_bool)
15
```

Shell

```
>>> %Run bool.py

  The result from 6 == 8 is False
  The result from 6 == 6 is True

>>>
```

Not Equals Operator !=

- The Python not equals operator, !=, is used to compare two values, variables or expressions to determine if they are **NOT** the same.
- If they are **NOT** the same, the operator returns **True**.
- If they **are the same**, then it returns **False**.

bool.py

```
colour1 = "Red"
colour2 = "Blue"
colour3 = "Red"

my_bool = colour1 != colour2

print("The result from Red != Blue is",my_bool

my_bool = colour1 != colour3

print("The result from Red != Red is",my_bool)
```

Shell

```
>>> %Run bool.py

  The result from Red != Blue is True
  The result from Red != Red is False

>>>
```

Relational Operators:
Equals and Not Equals

- Relational operators compare two items and return either True or False.
- two relational operators are:

  Equals: **==**

  Not equals: **!=**

- The operator takes the data type into account when making the comparison so a value of **10** would **NOT** be equal to the string value **"10"**.

Comparison Operators

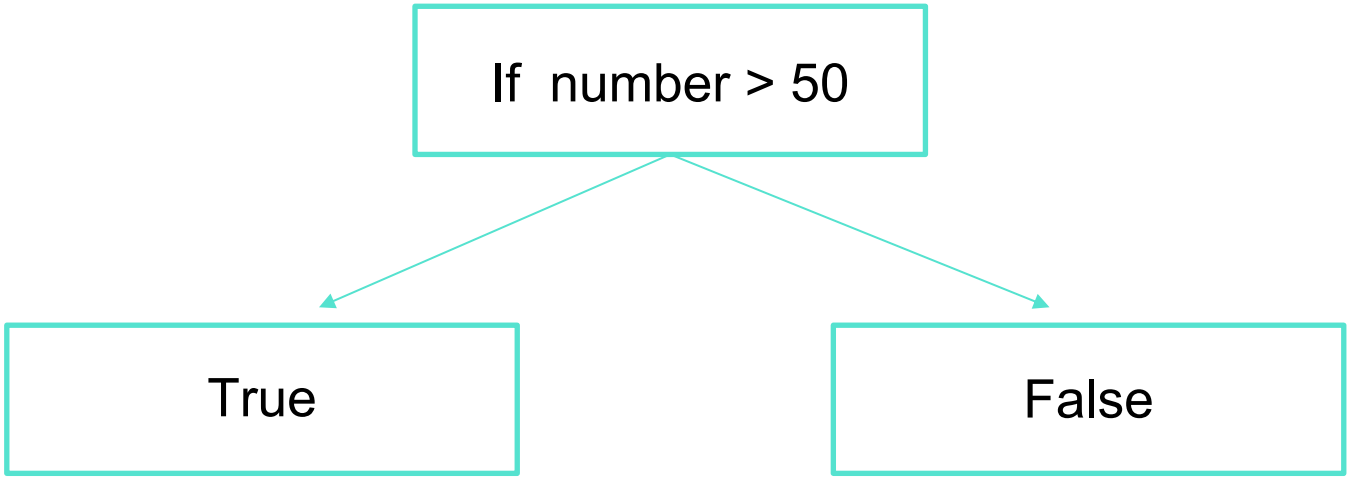
- In Python, *relational operators* compare two values or expressions. The most common ones are:
  - < less than
  - > greater than
  - <= less than or equal to
  - >= greater than or equal too

- If the relation is sound, then the entire expression will evaluate to True. If not, the expression evaluates to False.

bool.py | test.py *

```
1  a = 2
2  b = 3
3  a < b  # evaluates to True
4  a > b  # evaluates to False
5  a >= b # evaluates to False
6  a <= b # evaluates to True
7  a <= a # evaluates to True
8
```

Control Flow



A program's control flow is the order in which the program's code executes. The control flow of a Python program is regulated by conditional statements, loops, and function calls.

if Statement

- The **if** statement is used to determine the execution of code based on the evaluation of a Boolean expression.
- If the **if** statement expression evaluates to **True**, then the indented code following the statement is executed.
- If the expression evaluates to **False** then the **indented code** following the **if** statement is skipped and the program executes the **next line of code which is indented at the same level as the if statement**.

bool.py | test.py | Run current script

```
1
2
3  # if Statement
4
5  test_value = 100
6
7  if test_value > 1:
8    # Expression evaluates to True
9    print("This code is executed!")
10
11 if test_value > 1000:
12   # Expression evaluates to False
13   print("This code is NOT executed!")
14
15 print("Program continues at this point.")
16
```

Shell

```
>>> %Run test.py

  This code is executed!
  Program continues at this point.

>>>
```

Task

- Ask a user to input a number between 1 - 10
- Calculate if the number is above 5
- Write the results to the shell – i.e. This number is greater than/less than 5

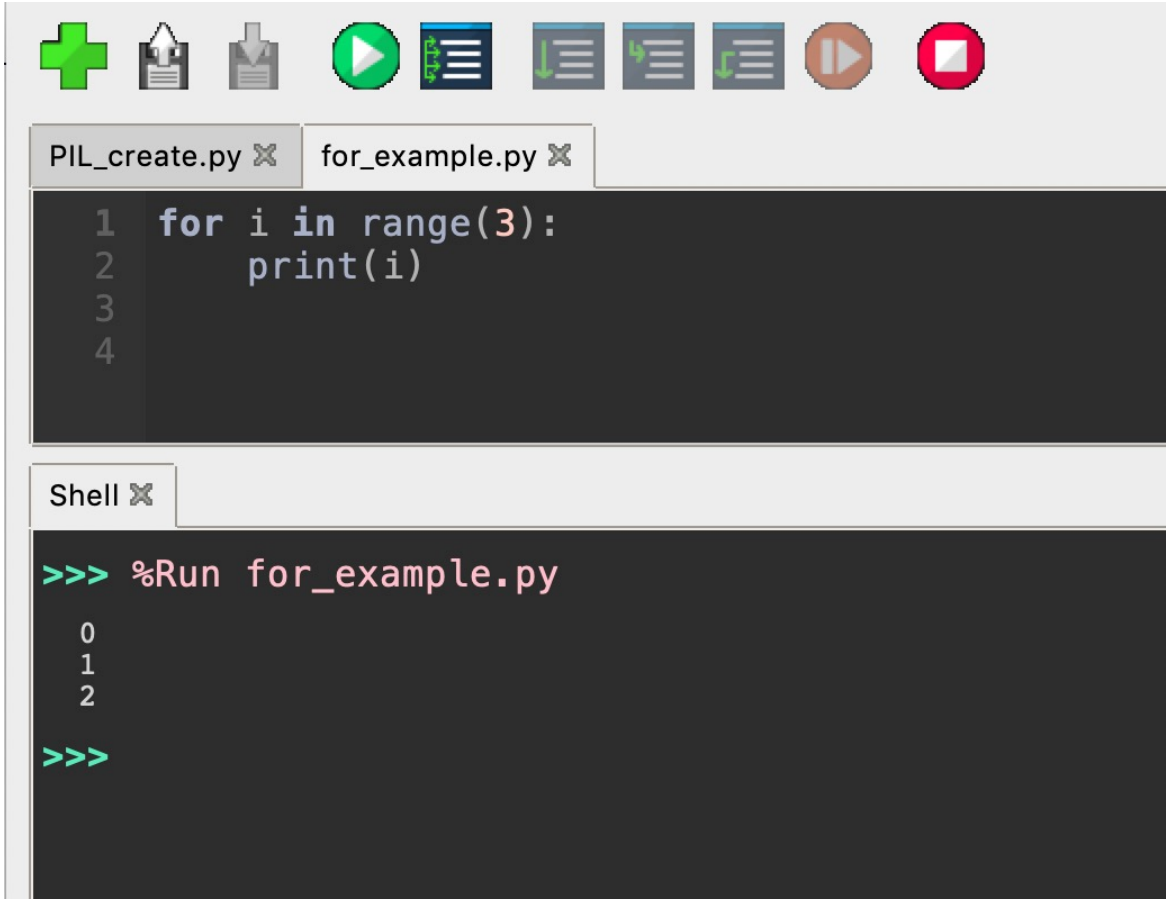
Tip

To get a number we write

```
user_num = int(input(“Choose a number between 1 – 10”))
```

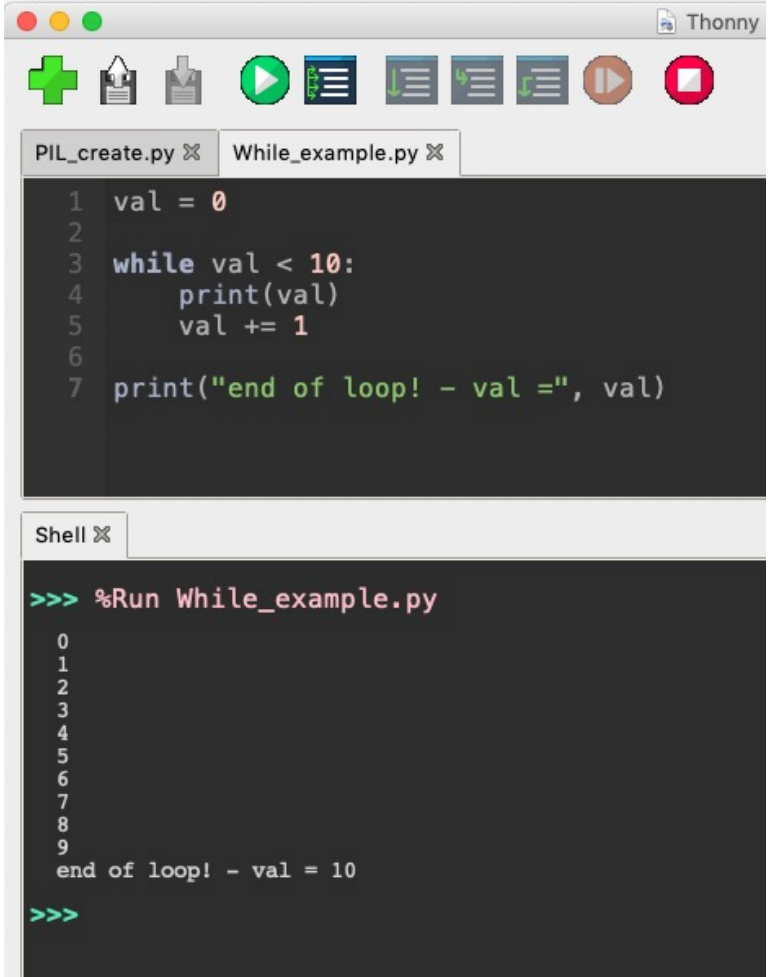
For

- In Python, a for loop can be used to perform an action a specific number of times in a row.
- The range() function can be used to create a list that can be used to specify the number of iterations in a for loop



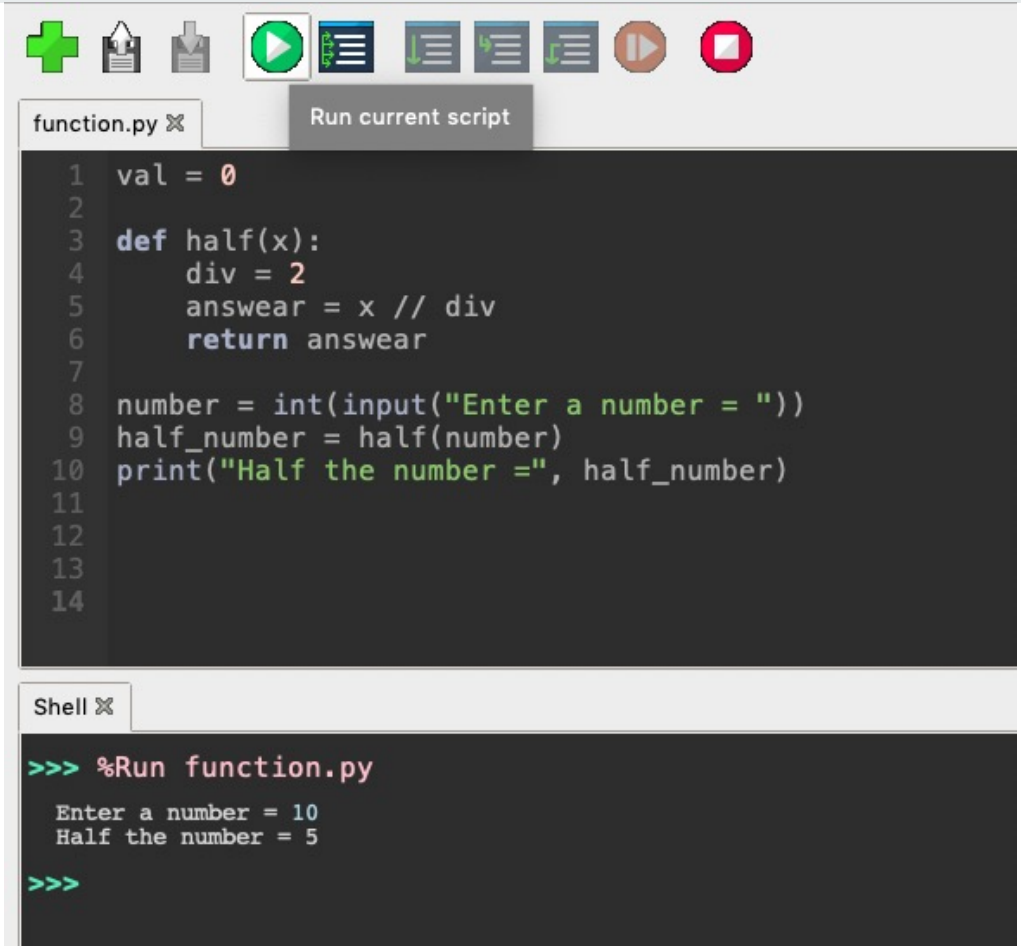
While

- In Python, a while loop will repeatedly execute a code block as long as a condition evaluates to **True**.
- The condition of a while loop is always checked first before the block of code runs. If the condition is not met initially, then the code block will never run.



Concept - Function

- In programming, as we start to write bigger and more complex programs, one thing we will start to notice is we will often have to repeat the same set of steps in many different places in our program.
- Functions are a convenient way to group our code into reusable blocks. A function contains a sequence of steps that can be performed repeatedly throughout a program without having to repeat the process of writing the same code again.



Using Libraries

- In Python, libraries are defined as a package or collection of various modules, which includes various functions or methods in modules that are imported into the program to perform some task without writing the large code snippets in the program.

eg

- **Pandas** - This is one of the open-source Python libraries which is mainly used in Data Science and machine learning subjects.
- **Numpy** - NumPy is another library that is used for mathematical functions. This library is mostly used in machine learning computations.
- **OpenCV -** is a library of programming functions mainly aimed at real-time computer vision.

- In Thonny you can add Libraries from **Tools** -> **Manage Packages**

Pillow

PIL is the Python Imaging Library.

See: - https://pillow.readthedocs.io/en/stable/handbook/index.html

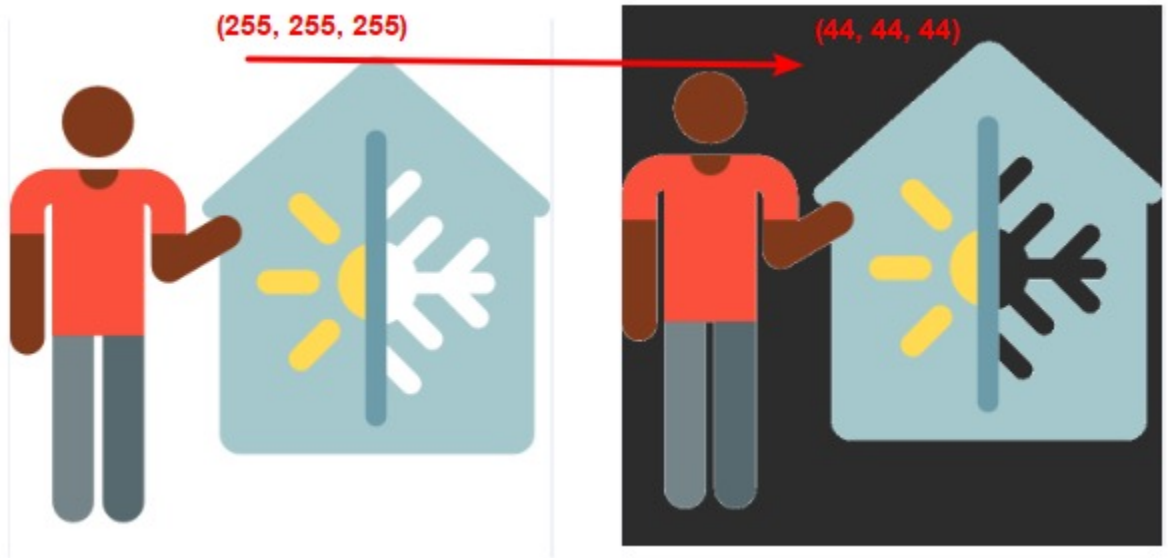
Example

PIL_create.py

```
1  from PIL import Image
2  import numpy as np
3  img = Image.open("file.png")
4
5  print(img.mode) #RGB
6  print(img.size)
7
8  width = img.size[0]
9  height = img.size[1]
10 for i in range(0,width):# process all pixels
11     for j in range(0,height):
12         data = img.getpixel((i,j))
13         #print(data) #(255, 255, 255)
14         if (data[0]==255 and data[1]==255 and data[2]==255):
15             img.putpixel((i,j),(44, 44, 44))
16 img.show()
```

Shell

```
>>> %Run PIL_create.py

  RGBA
  (271, 276)

>>>
```


(255, 255, 255)
(44, 44, 44)

Task –

- Use your own image
- Change colours using Pillow
- Look at the online Pillow Handbook - https://pillow.readthedocs.io/en/stable/handbook/index.html
- Can you find any other image transformation

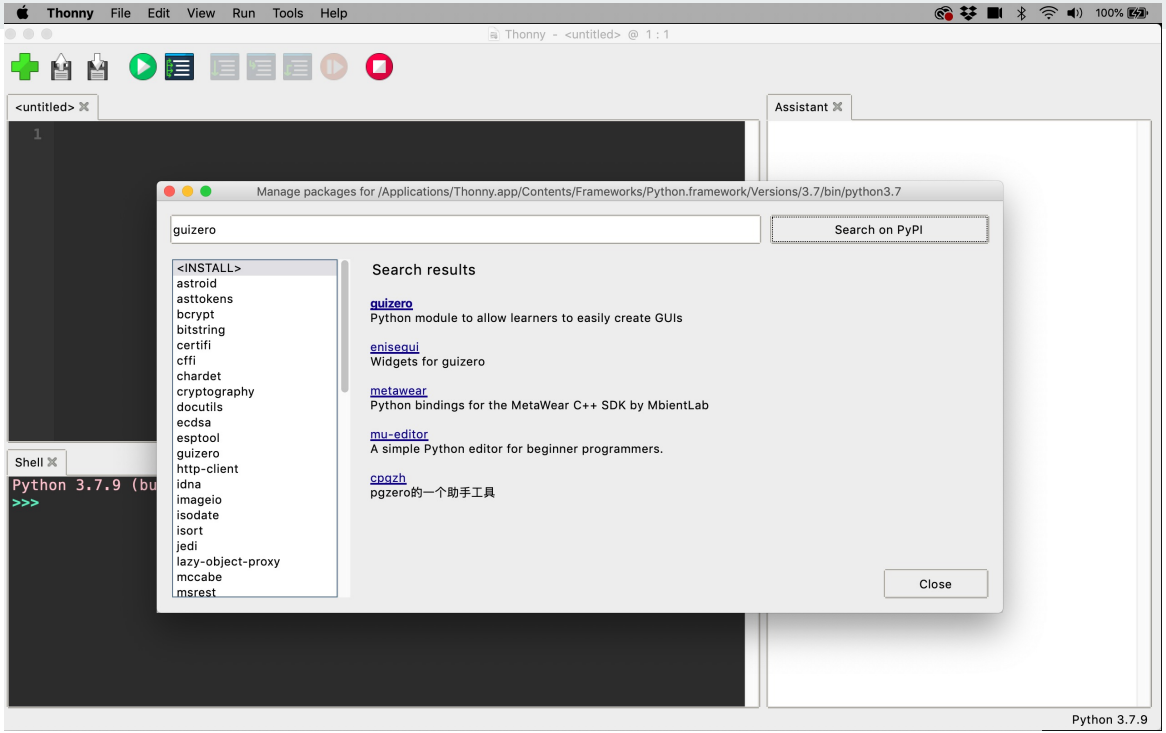
Conclusion

z

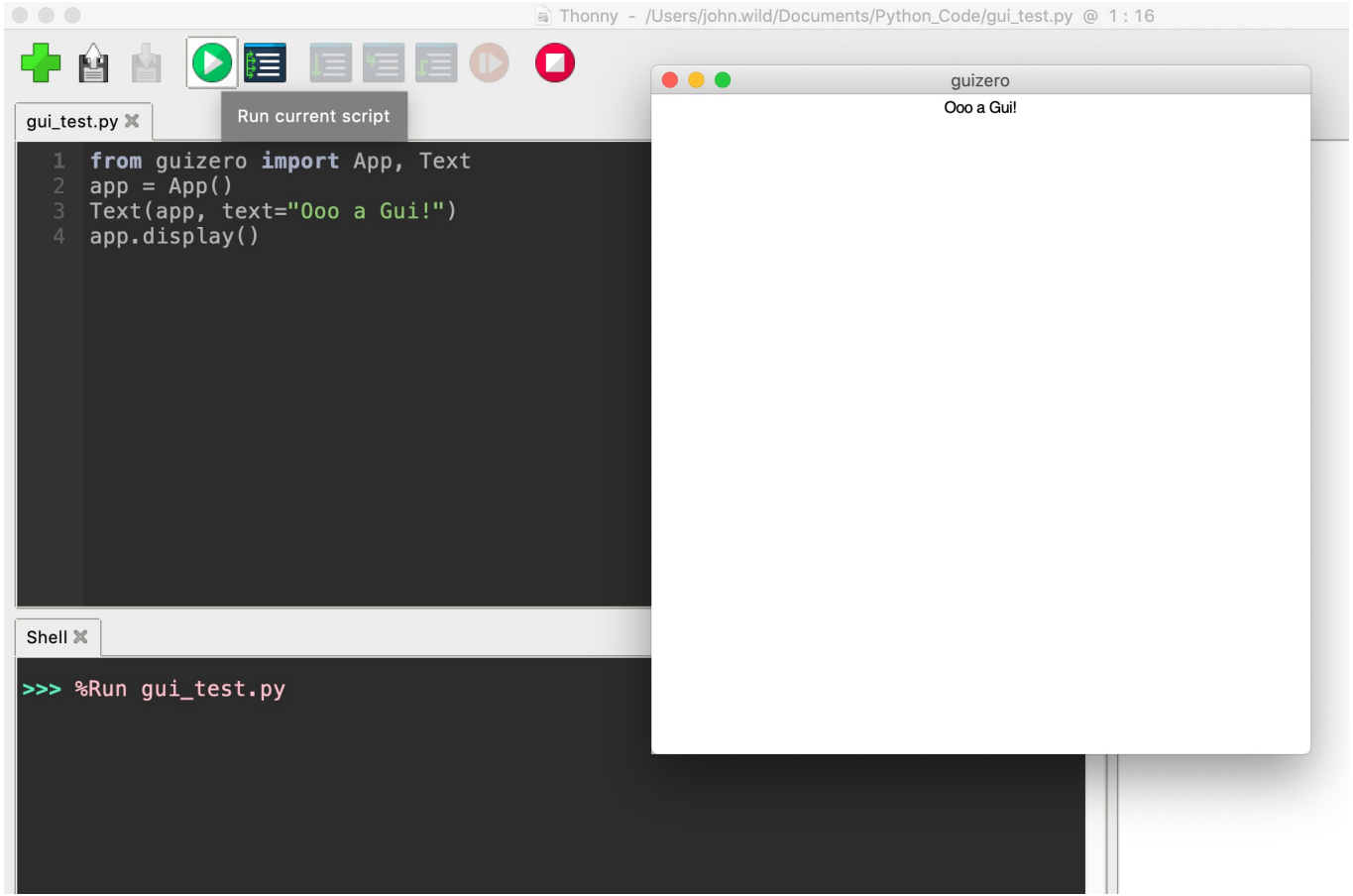
Extras .....

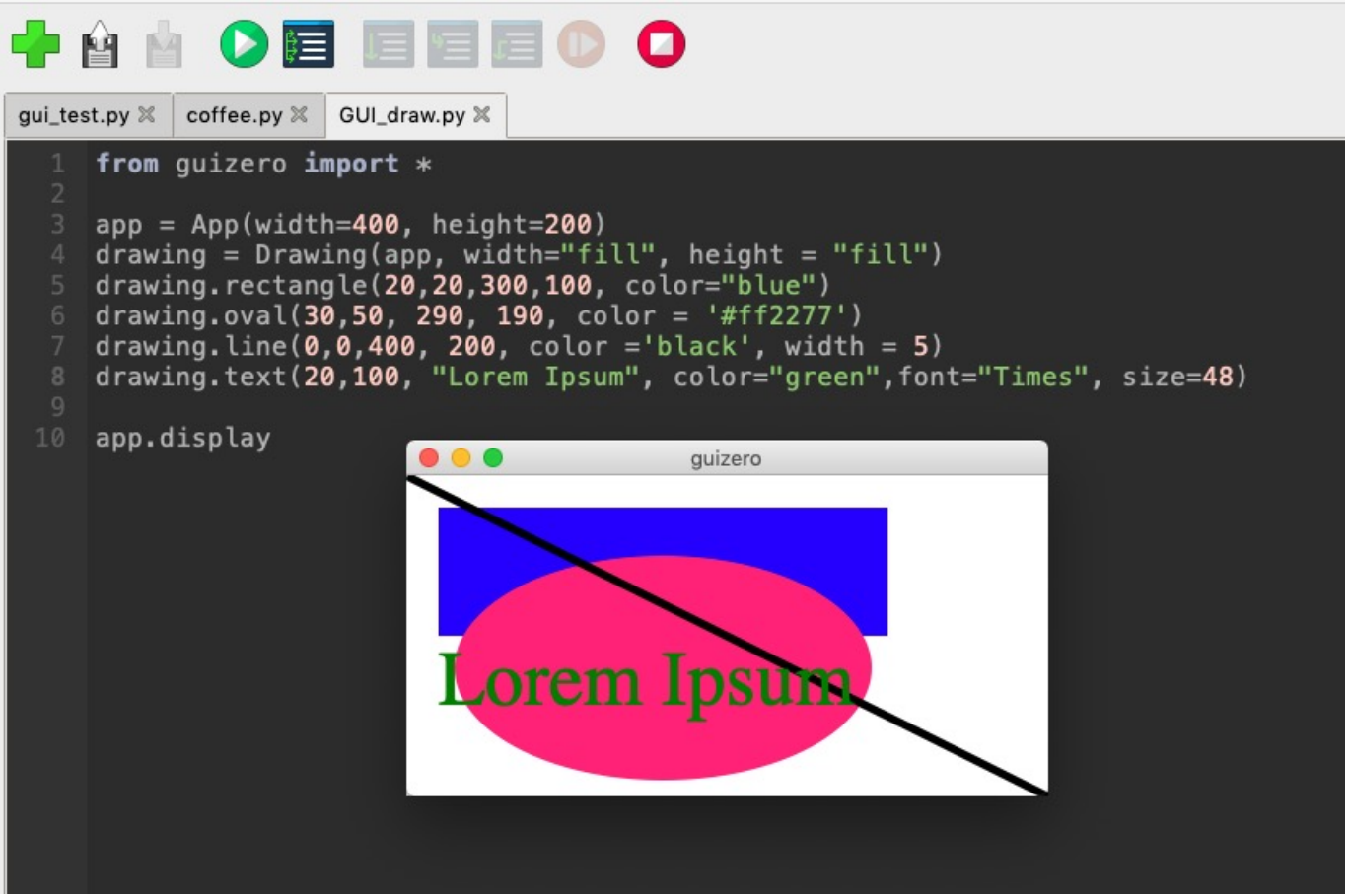
Guizero

Task – Install guizero



- **guizero** is a Python 3 library for creating simple GUIs
- open **Tools** -> **Manage Packages**

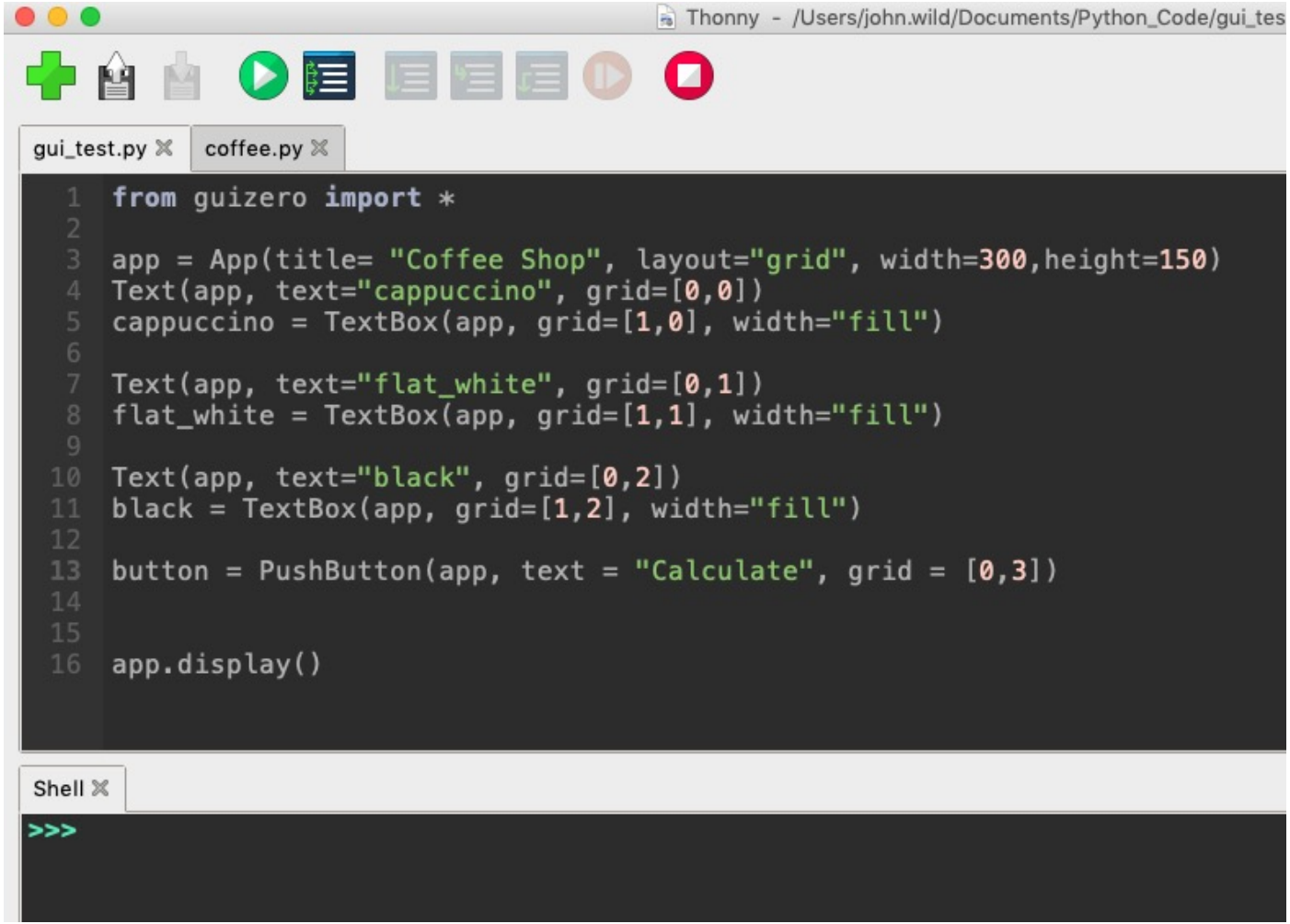

Thonny - /Users/john.wild/Documents/Python_Code/gui_test.py @ 1 : 16
Run current script
gui_test.py
1 from guizero import App, Text
2 app = App()
3 Text(app, text="Ooo a Gui!")
4 app.display()
Shell
>>> %Run gui_test.py
guizero
Ooo a Gui!



Task

1) read - Uning Guizero at: https://lawsie.github.io/guizero/
2) Experiment withyour own guizero

Getting User Input



Bringing it all together

gui_test.py | coffee.py

```
from guizero import *

def calculate_coffee():
    cappuccino_cost = 2.5
    flat_white_cost = 2.0
    black_cost = 1.5

    total.value = (int(cappuccino.value)*cappuccino_cost) + (int(flat_white.value)*flat_white_cost) + (int(black.value)*black_cost)
    print(total)

app = App(title= "Coffee Shop", layout="grid", width=300,height=150)
Text(app, text="cappuccino - £2.5", grid=[0,0])
cappuccino = TextBox(app, grid=[1,0], width="fill")

Text(app, text="flat_white - £2.0", grid=[0,1])
flat_white = TextBox(app, grid=[1,1], width="fill")

Text(app, text="black - £1.5", grid=[0,2])
black = TextBox(app, grid=[1,2], width="fill")

Text(app, text="Cost = ", grid=[0,3])
total = Text(app, grid=[1,3])

button = PushButton(app, text = "Calculate", grid = [0,4], command=calculate_coffee)


app.display()
```